Modularity and computation in semantic theory

Simon Charlow (Rutgers)

Princeton University November 29, 2017

simoncharlow.com/princeton.pdf

Today

A bit about what **semanticists** do, and why natural language makes our jobs hard.

There's a rich menagerie of *kinds of meanings*, each of which requires us to enrich and revise our basic assumptions about what meanings are (or can be).

More specifically

Semanticists tend to respond to the menagerie by lexically and compositionally *generalizing to the worst case*. One-size-fits-all. This gets out of hand, posing problems for the language learner and the theorist.

Functional programmers instead look for repeated patterns, and abstract those out as separate, modular pieces (helper functions). These can be invoked *as needed* online in composition. This strategy has theoretical and empirical virtues.

Compositionality

What is it to know a language?

You'll need to know something about its **syntax** — which hierarchically ordered bits of structure count as part of $\mathcal{L}$? What are the things in $\mathcal{L}$, and how are they built?

More central to our goals today, you'll need to know something about $\mathcal{L}$'s **semantics** — how structures in $\mathcal{L}$ are systematically associated with *interpretations* by its speakers, such that $\mathcal{L}$ is a useful medium for *communication*.

Two ways syntax matters

Only some strings of words are recognizably part of (e.g.) English:

1. Matt devoured the donut.
2. *Matt donut the devoured.
3. *Matt devoured the donut Mary.

Two ways syntax matters

Only some strings of words are recognizably part of (e.g.) English:

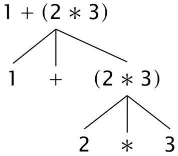
1. Matt devoured the donut.
2. *Matt donut the devoured.
3. *Matt devoured the donut Mary.

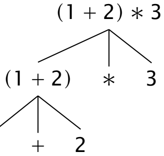
And some strings can be understood in multiple ways:

4. I saw the kestrel with the binoculars.

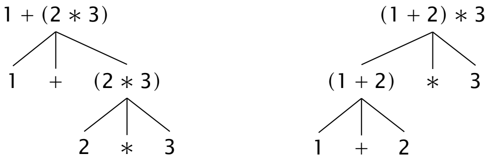
$1 + 2 * 3 = \ldots$

$1 + 2 * 3 = \ldots$





$1 + 2 * 3 = \ldots$



Porting the example to Haskell (result: an *ambiguous* and *recursive* grammar)...

```
data Term = Con Int | Term :+: Term | Term :*: Term

exp1 = Con 1 :+: (Con 2 :*: Con 3)  -- exp1 :: Term
exp2 = (Con 1 :+: Con 2) :*: Con 3  -- exp2 :: Term
-- Con 1 :+: Con 2 Con 3 yields an error !
-- As does Con 1 :+: (:*: Con 2 Con 3)   !
```

A semantics for our arithmetic language

```
data Term = Con Int | Term :+: Term | Term :*: Term

exp1 = Con 1 :+: (Con 2 :*: Con 3)
exp2 = (Con 1 :+: Con 2) :*: Con 3
```

A semantics for our arithmetic language

```haskell
data Term = Con Int | Term :+: Term | Term :*: Term

exp1 = Con 1 :+: (Con 2 :*: Con 3)
exp2 = (Con 1 :+: Con 2) :*: Con 3

eval (Con x)    = x
eval (a :+: b) = (eval a) + (eval b)
eval (a :*: b) = (eval a) * (eval b)
  -- eval exp1 = 7
  -- eval exp2 = 9
```

Types and (higher-order) functions

My interpreter says the following about the addition operation:

```
GHCi> :t (+)
(+) :: Int -> Int -> Int
```

Types and (higher-order) functions

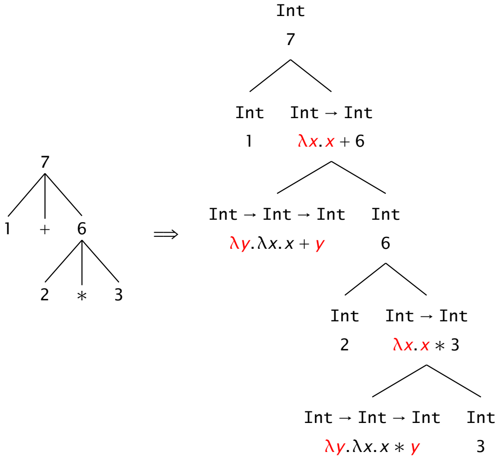
My interpreter says the following about the addition operation:

```
GHCi> :t (+)
(+) :: Int -> Int -> Int
```

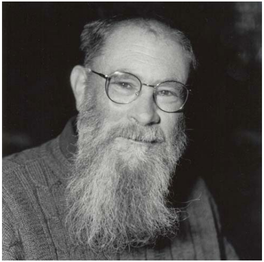
This is telling me that the **type** of `(+)` is such that it needs one `Int`, and then another, in order to produce an `Int`.

- So `(+)` is a **function** — a recipe for turning inputs to outputs — and it takes its inputs *one at a time*, making it **higher-order**.
- Functions represented with λ-calculus: if $f(x) = x^2$, we write f as $\lambda x.x^2$.

Interpretation as iterative function application

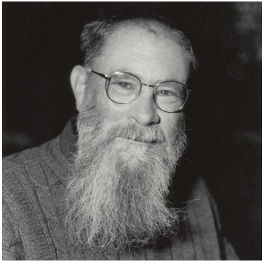


What are *linguistic* meanings?



Lewis (1970): "In order to say what a meaning *is*, we may first ask what a meaning *does*, and then find something that does that."

What are *linguistic* meanings?



Lewis (1970): "In order to say what a meaning *is*, we may first ask what a meaning *does*, and then find something that does that."

What *do* meanings do? Lewis was interested in the meanings of simple declarative sentences like *Porky grunts* (it was early days!). Sentences like that present the world as being a certain way (i.e., Porky's a grunter).

Knowing a (declarative) sentence meaning is knowing (a recipe for determining) whether that sentence is **True** or **False**.

A baseline extensional semantic theory

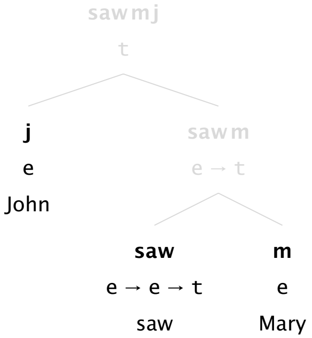
Inductively define the space of possible meanings, sorted by type:

$$\tau ::= e \mid t \mid \underbrace{\tau \to \tau}_{e\to t,\ (e\to t)\to t,\ \ldots}$$

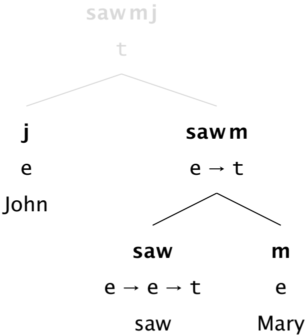
Interpret binary combination via (type-driven) functional application:

$$[\![\alpha\ \beta]\!] := [\![\alpha]\!][\![\beta]\!] \text{ or } [\![\beta]\!][\![\alpha]\!], \text{ whichever is defined}$$

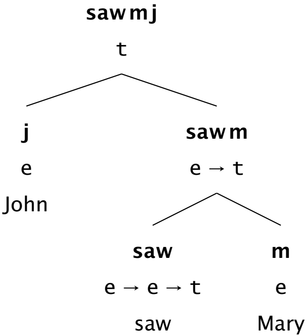
A sample derivation



A sample derivation



A sample derivation



Extending

Setting the stage

This picture is awesome. But a lot of important stuff doesn't fit neatly in it.

- Indexicality and pronouns
- Questions (i.e., as sets of propositions)
- Focus (and association with focus)
- Supplemental content (and projection)
- Quantification (and scope-taking)
- ...

Indexicality and pronouns

Indexical and pronominal expressions are chameleons: their values shift with the context of utterance — and sometimes multiple times within a single sentence!

1. John saw **me**.
2. It's cold **here now**!
3. John saw **her**.
4. Every philosopher$_i$ thinks **they**$_i$'re a genius.

The usual approach: indexicals and pronouns depend on the context of utterance.

$$\lambda c.\,\mathbf{get}_c$$

Implementing context-sensitivity

Extend the baseline theory such that meanings **uniformly depend on contexts**:

$$\tau_\circ ::= e \mid t \mid \tau_\circ \to \tau_\circ \qquad \tau ::= \underbrace{c \to \tau_\circ}_{c \to e \to t,\ c \to (e \to t) \to t,\ \ldots}$$

Interpret composition as **context-sensitive** functional application:

$$[\![\alpha\ \beta]\!] := \lambda c. [\![\alpha]\!]\, c\, ([\![\beta]\!]\, c)$$

Implementing context-sensitivity

Extend the baseline theory such that meanings **uniformly depend on contexts**:

$$\tau_\circ ::= e \mid t \mid \tau_\circ \to \tau_\circ \qquad \tau ::= \underbrace{c \to \tau_\circ}_{c \to e \to t,\; c \to (e \to t) \to t,\; \ldots}$$

Interpret composition as **context-sensitive** functional application:

$$[\![\alpha\ \beta]\!] := \lambda c. [\![\alpha]\!]\, c\, ([\![\beta]\!]\, c)$$

Sample derivation

$\lambda c.\,\mathbf{saw}\,\mathbf{get}_c\,\mathbf{j}$

$c \to t$

$\lambda c.\mathbf{j}$

$c \to e$

John

$\lambda c.\,\mathbf{saw}\,\mathbf{get}_c$

$c \to e \to t$

$\lambda c.\,\mathbf{saw}$

$c \to e \to e \to t$

saw

$\lambda c.\,\mathbf{get}_c$

$c \to e$

her

The basic intuition: do function application "inside the $c \to$".

Sample derivation

$\lambda c.\,\mathbf{saw}\,\mathbf{get}_c\,\mathbf{j}$

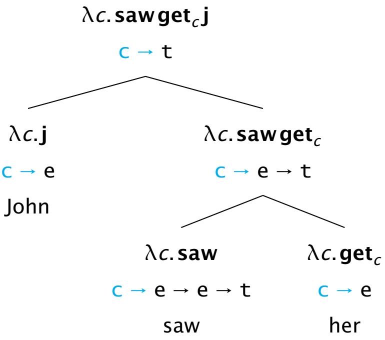
$c \to t$

$\lambda c.\mathbf{j}$ | $\lambda c.\,\mathbf{saw}\,\mathbf{get}_c$
$c \to e$ | $c \to e \to t$
John |

$\lambda c.\,\mathbf{saw}$ | $\lambda c.\,\mathbf{get}_c$
$c \to e \to e \to t$ | $c \to e$
saw | her

The basic intuition: do function application "inside the $c \to$".

Sample derivation



The basic intuition: do function application "inside the $c \to$".

Questions: intuition

A common approach to question semantics (Hamblin 1973, Karttunen 1977) treats questions as denoting *sets of their possible answers*:

$$[\![\text{who did John see?}]\!] = \{\mathbf{saw}\,x\,\mathbf{j} \mid \mathbf{human}\,x\}$$

And a common approach to deriving sets of answers is to begin by treating $[\![\text{who}]\!]$ as a set of *alternatives*, $\{x \mid \mathbf{human}\,x\}$.

Like pronouns, we have an immediate compositional challenge: how to compose sets of alternatives to yield bigger sets of alternatives?

Alternative semantics (Hamblin 1973)

Extend the baseline extensional theory such that the compositionally relevant meanings are **uniformly sets of alternatives**:

$$\tau_\circ ::= e \mid t \mid \tau_\circ \to \tau_\circ \qquad \tau ::= \underbrace{S\,\tau_\circ}_{S\,(e\to t),\; S\,((e\to t)\to t),\;\ldots}$$

Interpret binary combination via **alternative-friendly** functional application:

$$[\![\alpha\ \beta]\!] := \{f\,x \mid f \in [\![\alpha]\!],\, x \in [\![\beta]\!]\}$$

Sample derivation

$\{\mathbf{met}\, x\, \mathbf{j} \mid \mathbf{human}\, x\}$

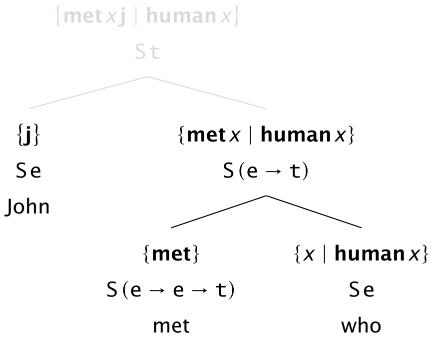
$\mathtt{S}\,\mathtt{t}$

$\{\mathbf{j}\}$	$\{\mathbf{met}\, x \mid \mathbf{human}\, x\}$
$\mathtt{S}\,\mathtt{e}$	$\mathtt{S}\,(\mathtt{e} \to \mathtt{t})$
John

$\{\mathbf{met}\}$	$\{x \mid \mathbf{human}\, x\}$
$\mathtt{S}\,(\mathtt{e} \to \mathtt{e} \to \mathtt{t})$	$\mathtt{S}\,\mathtt{e}$
met	who

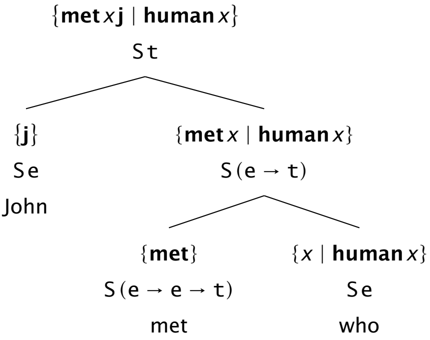
The basic intuition: do function application “inside the S”.

Sample derivation



The basic intuition: do function application “inside the S”.

Sample derivation



The basic intuition: do function application "inside the S".

Supplementation

In **apposition** and **presupposition**, certain aspects of meaning appear to be semantically *independent* of the rest of an utterance:

1. John hasn't finished *Burr*, **which is by Gore Vidal**.
2. If **the escalator in Sayles** is broken, I'll be upset.

Supplementation

In **apposition** and **presupposition**, certain aspects of meaning appear to be semantically *independent* of the rest of an utterance:

1. John hasn't finished *Burr*, **which is by Gore Vidal**.
2. If **the escalator in Sayles** is broken, I'll be upset.

A standard approach to these phenomena takes meanings to be *two-dimensional*, a pair of a "regular" value and some independent fact(s):

$$(\mathbf{b}, \mathbf{by\,gv\,b}) \qquad (\mathbf{the.esc}, \mathbf{theres.an.esc})$$

With a concomitant enrichment of $[\![\cdot]\!]$...

Association with focus and scalar implicature

In **association with focus** and **scalar implicature**, we consider an utterance, alongside other alternative utterances its speaker might have proffered:

1. I only introduced JOHN to Mary.
   $\Longrightarrow$ I introduced John to Mary, and I didn't introduce anyone else to Mary.
2. I ate two cookies.
   $\Longrightarrow$ I ate two cookies, and I didn't eat three cookies.

Association with focus and scalar implicature

In **association with focus** and **scalar implicature**, we consider an utterance, alongside other alternative utterances its speaker might have proffered:

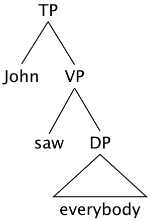
1. I only introduced JOHN to Mary.
   $\Longrightarrow$ I introduced John to Mary, and I didn't introduce anyone else to Mary.
2. I ate two cookies.
   $\Longrightarrow$ I ate two cookies, and I didn't eat three cookies.

A standard approach to these phenomena takes meanings to be *two-dimensional*, a pair of a "regular" value and alternative values (Rooth 1985, Sauerland 2004):

$$(\mathbf{j}, \{\mathbf{j}, \mathbf{k}, \mathbf{l}, \ldots\}) \qquad (\mathbf{2}, \{\mathbf{2}, \mathbf{3}, \mathbf{4}, \ldots\})$$

With a concomitant enrichment of $[\![\cdot]\!]$...

Scope-taking

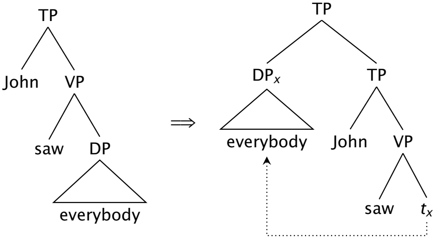


Some expressions need access to more than their immediate semantic context:

$$\textbf{everybody}\,(\lambda x.\,\textbf{saw}\,x\,\textbf{j})$$

Hm!

Is this generally handled in the same way as the other enrichments we've seen so far? Nope! Gets a sui generis treatment:



Applicatives

Lexical worries

For any given notion of enrichment, lexical entries for things that don't require that functionality need to be spuriously complicated:

$$\lambda c.\mathbf{j} \qquad \{\mathbf{j}\} \qquad \ldots$$

And you can't dine a la carte. If you want more than one enrichment, you'll need your lexical entries to *all* do justice to it. So for example, *everything* will need to be listed in the lexicon as context-sensitive, alternative-ready, supplement-compatible, focus-OK, quantification-copacetic, etc.

$$[\![\text{Matt}]\!] = ???$$

Learnability and semantic “cartography”

This poses obvious challenges for the language learner (to say nothing of the theorist). Every time some new notion of meaning is brought into view, the entire grammar (lexicon and composition rules) needs to be *rewritten from scratch*.

Moreover, the nature of the rewrite is in general under-determined by the data. Should we have context-dependent alternative sets, or sets of context-dependent meanings? Should we *have to* choose, absent any data?

Functional languages

Functional programming languages are, just like NL grammars in the Frege/Montague tradition, built on functions and arguments.

Just like NL semanticists, functional programmers frequently find themselves hungering for *effects*, and ways to systematically express concepts that lie outside the core features of their language.

Some common effects

- Environment (valuing variables in a global namespace)
- Nondeterminism (carrying out multiple computations in parallel)
- Pointed nondeterminism (flagging a particular value as central)
- Logging (keeping a side-log of execution-incidental info)
- Control (aborting a computation, jumping around inside a program)

These all correspond in a fairly direct way to things that are useful for doing natural language semantics!

Abstraction and modularity

Functional programmers are lazy. When they see a bit of code occurring over and over again in their program, they abstract it out as a separate function.

Let's see how far this gets us.

For context-dependence

The idea — almost embarrassing in its simplicity — is to just abstract out and modularize the core features of the standard account.

For context-dependence, the standard approach involves lifting the entire lexicon, and composing everything via $[\![\cdot]\!]$.

The modular alternative is to leave the lexicon as simple as possible, rely only on $[\![\cdot]\!]$, and invoke helper functions whenever we need to do something *fancier*.

More formally

In lieu of treating everything as trivially dependent on an context, invoke a function ρ which turns any x into a constant function from contexts into x:

$$\rho := \underbrace{\lambda x.\lambda c.\, x}_{a \to \mathrm{c} \to a}$$

More formally

In lieu of treating everything as trivially dependent on an context, invoke a function ρ which turns any x into a constant function from contexts into x:

$$\rho := \underbrace{\lambda x.\lambda c.\, x}_{a \to \mathsf{c} \to a}$$

Instead of taking on $[\![\cdot]\!]$ wholesale, we'll help ourselves to a function $\circledast$ which allows us to perform context-friendly function application on demand:

$$\circledast := \underbrace{\lambda m.\lambda n.\lambda c.\, m\,c\,(n\,c)}_{(\mathsf{c} \to a \to b) \to (\mathsf{c} \to a) \to \mathsf{c} \to b}$$

Sample derivations

$\lambda c.\, \mathbf{spoke}\, \mathbf{get}_c$

$\mathsf{c} \to \mathsf{t}$

$\lambda c.\, \mathbf{get}_c$

$\mathsf{c} \to \mathsf{e}$

she

$\lambda n.\lambda c.\, \mathbf{spoke}\,(n c)$

$(\mathsf{c} \to \mathsf{e}) \to \mathsf{c} \to \mathsf{t}$

$\circledast$

$\lambda c.\, \mathbf{spoke}$

$\mathsf{c} \to \mathsf{e} \to \mathsf{t}$

ρ

$\mathbf{spoke}$

$\mathsf{e} \to \mathsf{t}$

spoke

Sample derivations

$\lambda c.\,\mathbf{spoke}\,\mathbf{get}_c$

$c \to t$

$\lambda c.\,\mathbf{get}_c$

$c \to e$

she

$\lambda n.\lambda c.\,\mathbf{spoke}\,(n\,c)$

$(c \to e) \to c \to t$

$\circledast$

$\lambda c.\,\mathbf{spoke}$

$c \to e \to t$

ρ

$\mathbf{spoke}$

$e \to t$

spoke

Sample derivations

$\lambda c.\,\mathbf{spoke}\,\mathbf{get}_c$

$\mathtt{c} \to \mathtt{t}$

$\lambda c.\,\mathbf{get}_c$

$\mathtt{c} \to \mathtt{e}$

she

$\lambda n.\lambda c.\,\mathbf{spoke}\,(n\,c)$

$(\mathtt{c} \to \mathtt{e}) \to \mathtt{c} \to \mathtt{t}$

$\circledast$

$\lambda c.\,\mathbf{spoke}$

$\mathtt{c} \to \mathtt{e} \to \mathtt{t}$

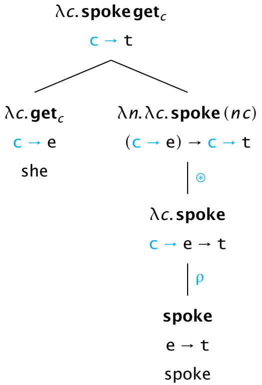
ρ

$\mathbf{spoke}$

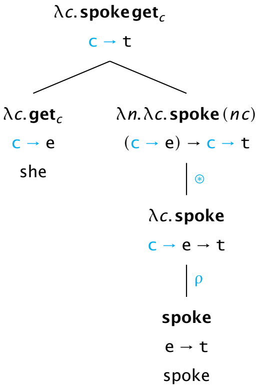
$\mathtt{e} \to \mathtt{t}$

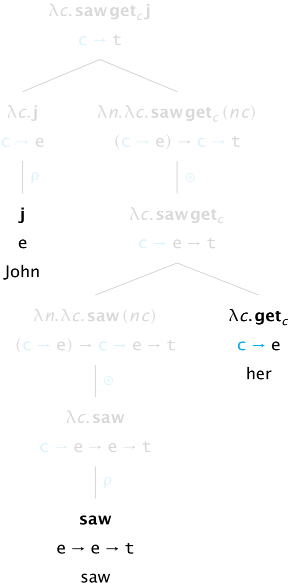
spoke

Sample derivations

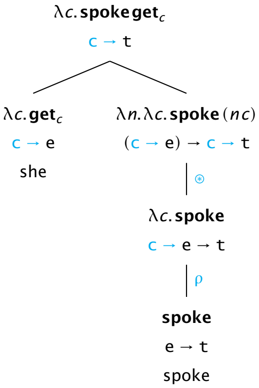


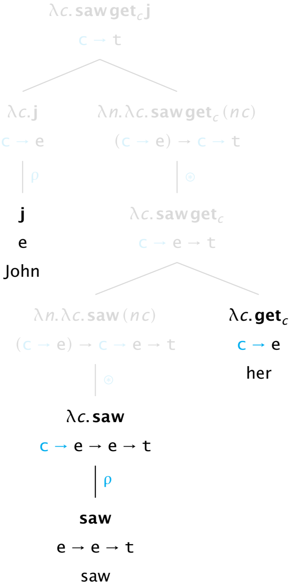
Sample derivations



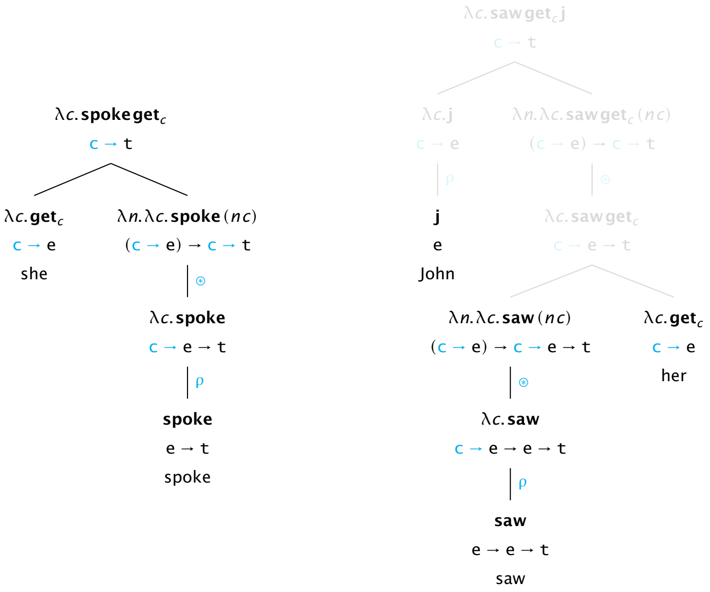


Sample derivations

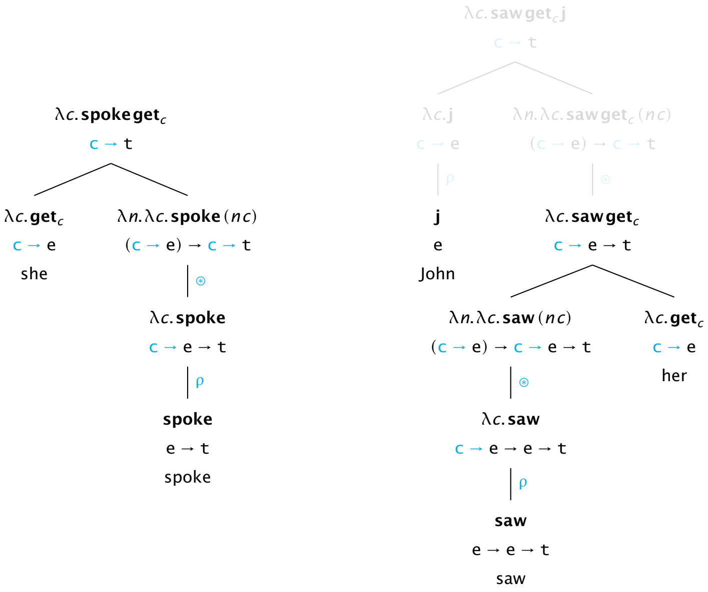




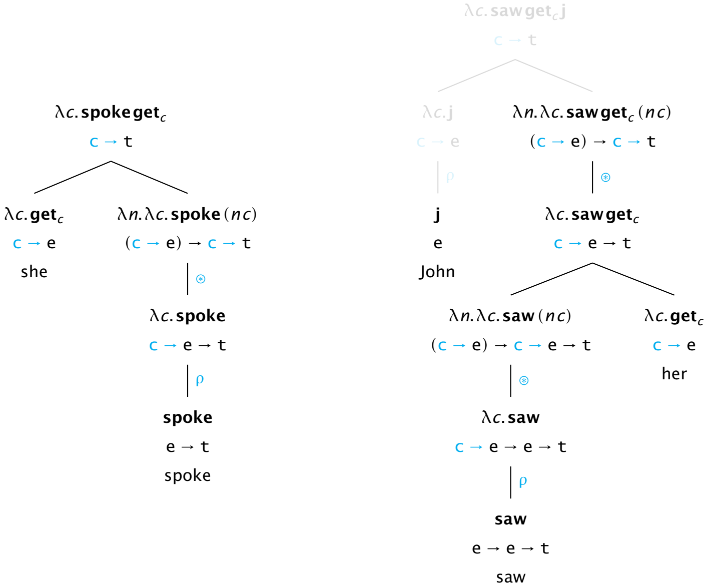
Sample derivations



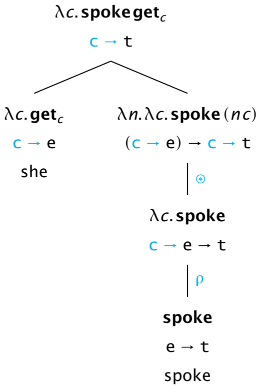
Sample derivations

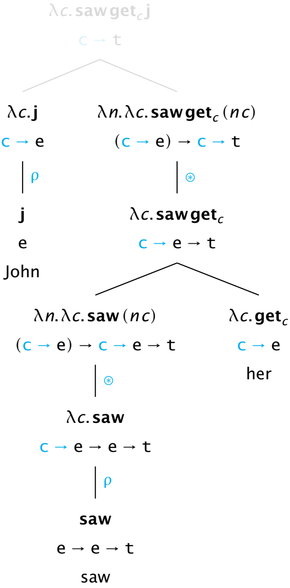


Sample derivations

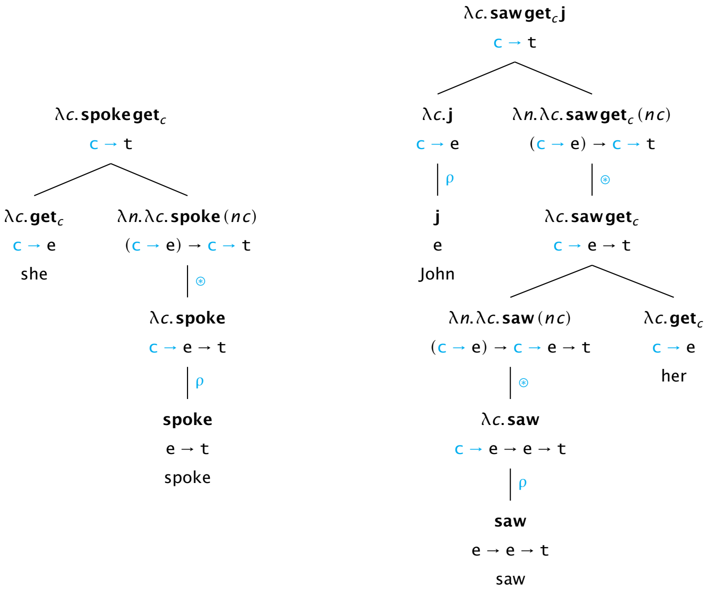


Sample derivations



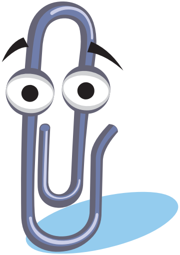


Sample derivations



Basically

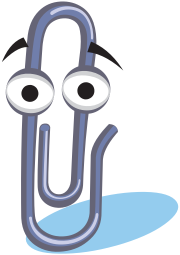
Basically



It looks like you're trying to do semantics.

Would you like help?

Basically

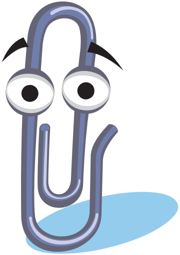


It looks like you're trying to do semantics.

Would you like help?

- ☐ Give me a ρ

Basically

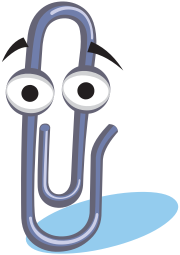


It looks like you're trying to do semantics.

Would you like help?

- ☐ Give me a ρ
- ☐ Give me a $\circledast$

Basically

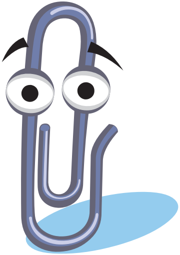


It looks like you're trying to do semantics.

Would you like help?

- ☐ Give me a ρ
- ☐ Give me a $\circledast$
- ☐ Don't show me this tip again

Basically



It looks like you're trying to do semantics.

Would you like help?

- ☐ Give me a ρ
- ☑ Give me a $\circledast$
- ☐ Don't show me this tip again

A familiar construct

When we abstract out ρ and $\circledast$ in this way, we're in the presence of something known to computer scientists and functional programmers as an **applicative functor** (McBride & Paterson 2008, Kiselyov 2015).

Applicative functors

Formally, an applicative functor is a type constructor F with two functions:

$$\rho :: a \to F\,a \qquad \circledast :: F\,(a \to b) \to F\,a \to F\,b$$

We might sum up the roles of these pieces as follows:

- F embodies some notion of “fanciness”, or enriched meaning.
- ρ tells you how to upgrade something into a trivially fancy thing.
- $\circledast$ characterizes a notion of *fancy combination* (e.g., function application).

Applicative laws

The ρ and $\circledast$ operations should satisfy some laws:

Homomorphism

$\rho\, f \circledast \rho\, x = \rho\, (f\, x)$

Identity

$\rho\, (\lambda x. x) \circledast v = v$

Interchange

$\rho\, (\lambda f. f\, x) \circledast u = u \circledast \rho\, x$

Composition

$\rho\, (\circ) \circledast u \circledast v \circledast w = u \circledast (v \circledast w)$

Basically, these laws guarantee that $\circledast$ is a kind of fancy functional application, and that ρ is a trivial way to make something fancy.

From GHC.Base (on Hackage)

```
class Functor f => Applicative f where
    pure  :: a -> f a
    (<*>) :: f (a -> b) -> f a -> f b
```

A new arithmetic interpreter

Using applicatives, we can rewrite `eval` in a way that works *for any applicative*:

```
eval :: Applicative f => Term -> f Int
eval (Con x)    = pure x
eval (a :+: b) = pure (+) <*> (eval a) <*> (eval b)
eval (a :*: b) = pure (*) <*> (eval a) <*> (eval b)
```

Generalizing the approach: alternatives

Another example of an applicative functor, for sets:

$$\rho\, x := \{x\} \qquad m \circledast n := \{\, f\, x \mid f \in m, x \in n \,\}$$

(See Charlow 2014, 2017 for more on this.)

Sample derivation, compared with context-sensitivity

$\{\textbf{spoke}\,x \mid x \in \textbf{human}\}$

$\mathtt{S\,t}$

$\{x \mid x \in \textbf{human}\}$	$\lambda n.\,\{\textbf{spoke}\,x \mid x \in n\}$
$\mathtt{S\,e}$	$\mathtt{S\,e \to S\,t}$
who	$\circledast$
	$\{\textbf{spoke}\}$
	$\mathtt{S\,(e \to t)}$
	ρ
	$\textbf{spoke}$
	$\mathtt{e \to t}$
	spoke

$\lambda c.\,\textbf{spoke}\,\textbf{get}_c$

$\mathtt{c \to t}$

$\lambda c.\,\textbf{get}_c$	$\lambda n.\lambda c.\,\textbf{spoke}\,(n\,c)$
$\mathtt{c \to e}$	$\mathtt{(c \to e) \to c \to t}$
she	$\circledast$
	$\lambda c.\,\textbf{spoke}$
	$\mathtt{c \to e \to t}$
	ρ
	$\textbf{spoke}$
	$\mathtt{e \to t}$
	spoke

Sample derivation, compared with context-sensitivity

$\{\mathbf{spoke}\,x \mid x \in \mathbf{human}\}$

$\mathtt{S\,t}$

$\{x \mid x \in \mathbf{human}\}$	$\lambda n.\,\{\mathbf{spoke}\,x \mid x \in n\}$
$\mathtt{S\,e}$	$\mathtt{S\,e} \to \mathtt{S\,t}$
who	$\circledast$
	$\{\mathbf{spoke}\}$
	$\mathtt{S\,(e \to t)}$
	ρ
	$\mathbf{spoke}$
	$\mathtt{e \to t}$
	spoke

$\lambda c.\,\mathbf{spoke}\,\mathbf{get}_c$

$\mathtt{c \to t}$

$\lambda c.\,\mathbf{get}_c$	$\lambda n.\lambda c.\,\mathbf{spoke}\,(n\,c)$
$\mathtt{c \to e}$	$\mathtt{(c \to e) \to c \to t}$
she	$\circledast$
	$\lambda c.\,\mathbf{spoke}$
	$\mathtt{c \to e \to t}$
	ρ
	$\mathbf{spoke}$
	$\mathtt{e \to t}$
	spoke

Sample derivation, compared with context-sensitivity

$\{\mathbf{spoke}\,x \mid x \in \mathbf{human}\}$

$\mathtt{S\,t}$

$\{x \mid x \in \mathbf{human}\}$ | $\lambda n.\,\{\mathbf{spoke}\,x \mid x \in n\}$
$\mathtt{S\,e}$ | $\mathtt{S\,e} \to \mathtt{S\,t}$
who | $\circledast$

$\{\mathbf{spoke}\}$

$\mathtt{S\,(e \to t)}$

ρ

$\mathbf{spoke}$

$\mathtt{e \to t}$

spoke

$\lambda c.\,\mathbf{spoke}\,\mathbf{get}_c$

$\mathtt{c \to t}$

$\lambda c.\,\mathbf{get}_c$ | $\lambda n.\lambda c.\,\mathbf{spoke}\,(n\,c)$
$\mathtt{c \to e}$ | $\mathtt{(c \to e) \to c \to t}$
she | $\circledast$

$\lambda c.\,\mathbf{spoke}$

$\mathtt{c \to e \to t}$

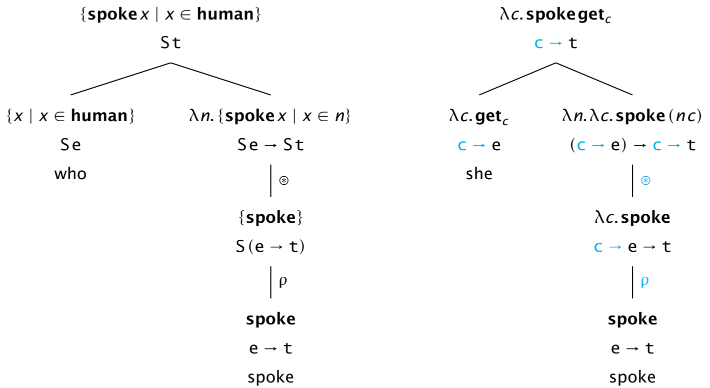
ρ

$\mathbf{spoke}$

$\mathtt{e \to t}$

spoke

Sample derivation, compared with context-sensitivity



Generality

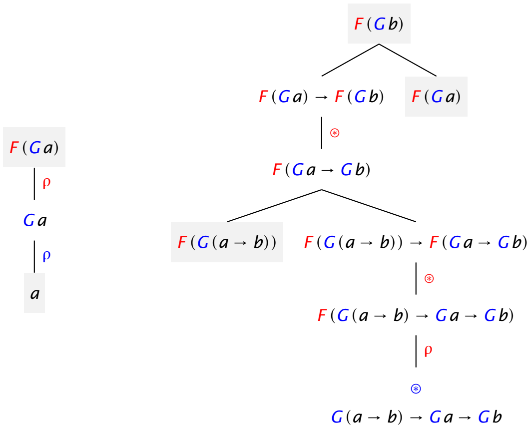
The technique is super general, and can be fruitfully applied (inter alia) to supplementation, (association with) focus, and scope:

$$\rho\, x \coloneqq (x, \mathbb{T}) \qquad (f, p) \circledast (x, q) \coloneqq (f\, x, p \wedge q)$$

$$\rho\, x \coloneqq (x, \{x\}) \qquad (f, F) \circledast (x, X) \coloneqq (f\, x, \{f'\, x' \mid f' \in F, x' \in X\})$$

$$\rho\, x \coloneqq \lambda k.\, k\, x \qquad m \circledast n \coloneqq \lambda k.\, m\, (\lambda f.\, n\, (\lambda x.\, k\, (f\, x)))$$

Applicative functors compose



Whenever you have two applicative functors, you're *guaranteed* to have two more!

Ross Paterson's Data.Functor.Compose (on Hackage)

```
module Data.Functor.Compose (
    Compose(..),
   ) where

newtype Compose f g a = Compose { getCompose :: f (g a) }

instance (Applicative f, Applicative g) =>
    Applicative (Compose f g) where
      pure x = Compose (pure (pure x))
      Compose f <*> Compose x = Compose ((<*>) <$> f <*> x)
```

Two examples

Composing context-sensitivity with sets of alternatives:

$$\rho\, x = \lambda c.\, \{x\} \qquad m \circledast n = \lambda c.\, \{f\, x \mid f \in m\, c,\ x \in n\, c\}$$

Composing sets of alternatives with context-sensitivity:

$$\rho\, x = \{\lambda c.\, x\} \qquad m \circledast n = \{\lambda c.\, f\, c\, (x\, c) \mid f \in m,\ x \in n\}$$

The lexicon

The lexicon remains maximally simple. Nothing needs to be listed as any fancier than it actually is. May simplify the learning problem.

No unprincipled across-the-board “cartographic” choices are required. Any ordering of effects can be generated in principle.

Charlow, Simon. 2014. *On the semantics of exceptional scope*. New York University Ph.D. thesis. http://semanticsarchive.net/Archive/2JmMWRjY/.

Charlow, Simon. 2017. The scope of alternatives: Indefiniteness and islands. Unpublished ms. http://ling.auf.net/lingbuzz/003302.

Hamblin, C. L. 1973. Questions in montague english. *Foundations of Language* 10(1). 41–53.

Karttunen, Lauri. 1977. Syntax and semantics of questions. *Linguistics and Philosophy* 1(1). 3–44. https://doi.org/10.1007/BF00351935.

Kiselyov, Oleg. 2015. Applicative abstract categorial grammars. In Makoto Kanazawa, Lawrence S. Moss & Valeria de Paiva (eds.), *NLCS'15. Third workshop on natural language and computer science*, vol. 32 (EPiC Series), 29–38.

Lewis, David. 1970. General semantics. *Synthese* 22(1-2). 18–67. https://doi.org/10.1007/BF00413598.

McBride, Conor & Ross Paterson. 2008. Applicative programming with effects. *Journal of Functional Programming* 18(1). 1–13. https://doi.org/10.1017/S0956796807006326.

Rooth, Mats. 1985. *Association with focus*. University of Massachusetts, Amherst Ph.D. thesis.

Sauerland, Uli. 2004. Scalar implicatures in complex sentences. *Linguistics and Philosophy* 27(3). 367–391. https://doi.org/10.1023/B:LING.0000023378.71748.db.